SpyWolf

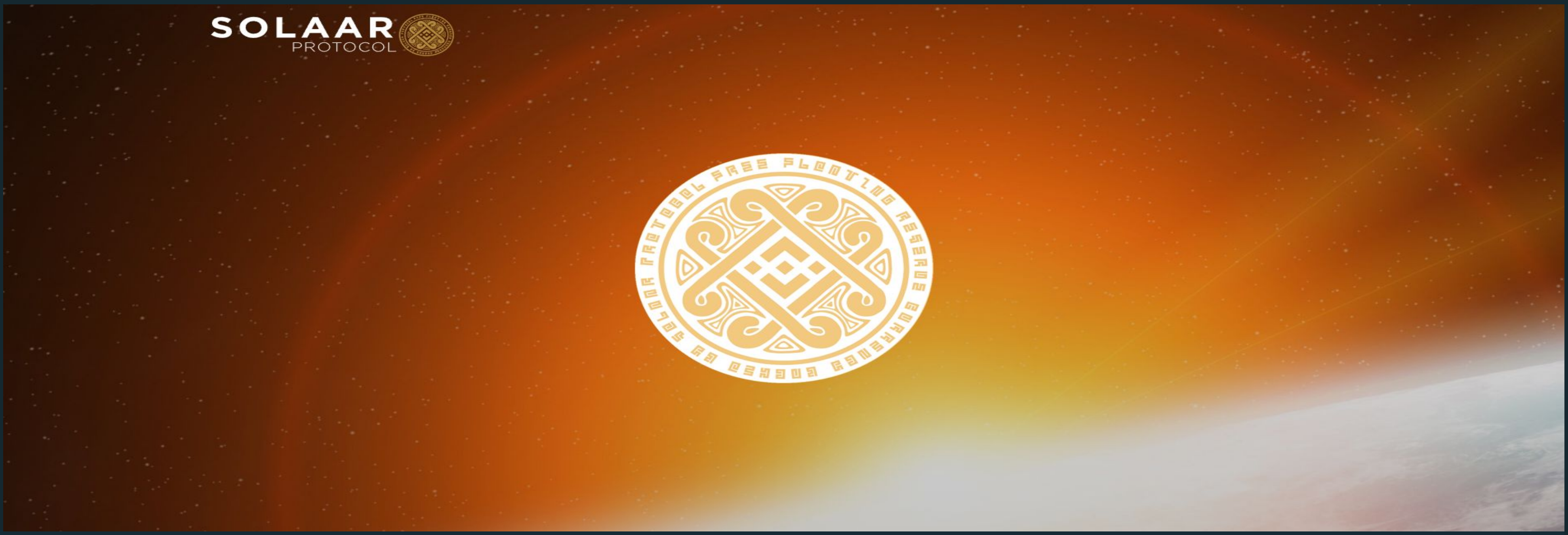
Project Audit

Not Deployed

Project:

SOLAAR

May 2, 2022



SOLAAR

Overview

This audit has been prepared for **SOLAAR** to review the main aspects of the project to help investors make an informative decision in the research process.

You will find a a summarized review of the following main key points:

- Contract's source code
- Project and team
- Website
- Social media & online presence

The following contracts are not deployed yet.

NOTE: We ONLY consider a project safe if they receive our "Certificate of Trust" NFT. This report only points out any potential red flags found in our analysis. Always do your own research before investing in a project.

Smart Contract Review

The contract review process pays special attention to the following:

- Testing the smart contracts against both common and uncommon vulnerabilities
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

```
const events = [
  'dragenter',
  'dragleave',
  'dragover', // to allow drop
  'drop'
];
events.forEach(e => {
  fileDropZone.addEventListener(e, (ev) => {
    ev.preventDefault();
    if (ev.type === 'dragenter') {
      fileDropZone.classList.add('solid-border'
    }
    if (ev.type === 'dragleave') {
      fileDropZone.classList.remove('solid-b
    }
    if(ev.type === 'drop') {
      fileDropZone.classList.remove('sol
      handleFiles(ev.dataTransfer.files
        .then(values => values.map(tag
          tag.setAttribute('class',
          fileDropZone.appendChild(
```

"The results of this audit are purely based on the team's evaluation and does not guarantee nor reflect the projects outcome and goal"
- SpyWolf Team

Smart Contract Summary

File name	**erc20.sol**
Contract Name	Assigned at deployment
Ticker	Assigned at deployment
Contract	N/A
Network	N/A
Language	Solidity
Tax	No tax
Total Supply	Minted after deployment
Status	Not deployed

Current stats

Burn	Token not deployed yet
LP Address	No liquidity added yet
Liquidity	No liquidity added yet
MaxTxAmount	No limit

Issues Checking Status	
Design Logic	Passed ✓
Compiler warnings.	Passed ✓
Private user data leaks	Passed ✓
Timestamp dependence	Passed ✓
Integer Overflow and Underflow	Passed ✓
Race conditions and Reentrancy. Cross-function race conditions	Passed ✓
Possible delays in data delivery	Passed ✓
Oracle calls	Passed ✓
Front running	Passed ✓
DoS with Revert	Passed ✓
DoS with block gas limit	Passed ✓
Methods execution permissions	Passed ✓
Economy model	Passed ✓
The impact of the exchange rate on the logic	Passed ✓
Malicious Event log	Passed ✓
Scoping and Declarations	Passed ✓
Uninitialized storage pointers	Passed ✓
Arithmetic accuracy	Passed ✓
Cross-function race conditions	Passed ✓
Safe Zeppelin module	Passed ✓
Fallback function security	Passed ✓

Featured Wallets

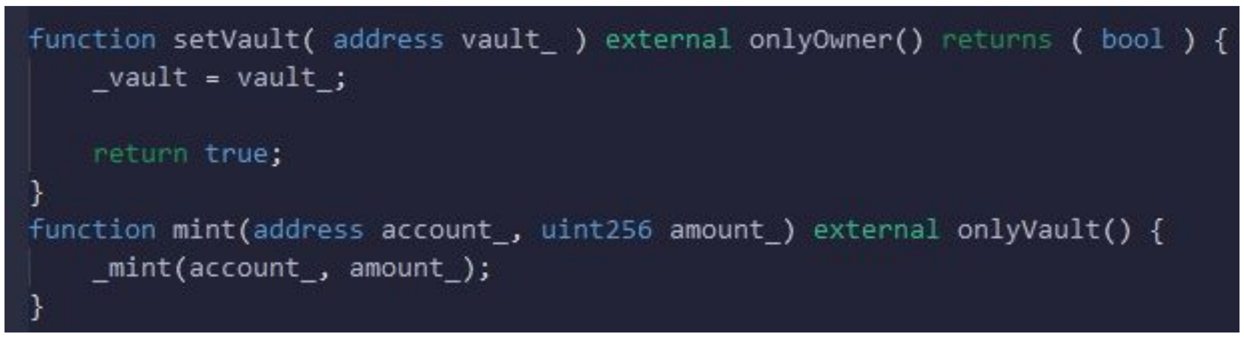
Owner address	N/A
LP token address	No liquidity added yet

Top 3 Unlocked Wallets

Wallet 1 (100%)	N/A

Security Threads

Owner can set vault address.
⚠️Vault address can mint new tokens.



```
function setVault( address vault_ ) external onlyOwner() returns ( bool ) {
    _vault = vault_;

    return true;
}
function mint(address account_, uint256 amount_) external onlyVault() {
    _mint(account_, amount_);
}
```

Smart Contract Summary

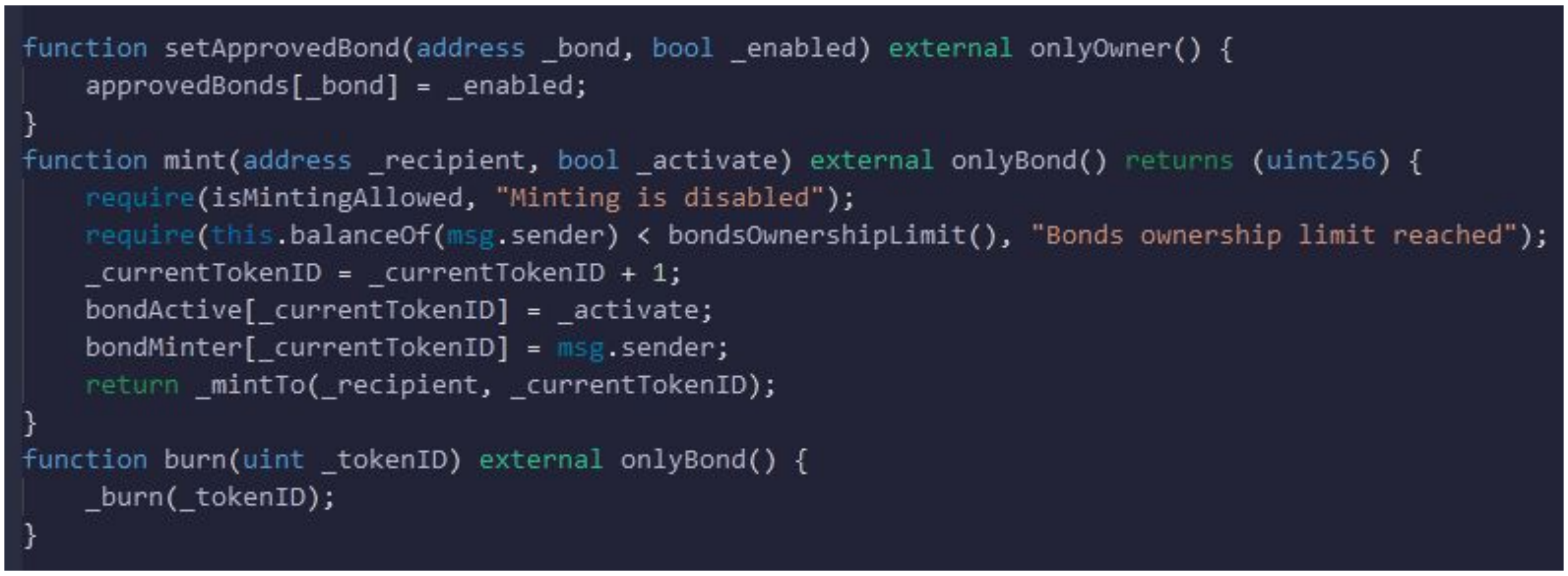
File name	**bond_nft.sol**
Contract Name	N/A
Ticker	N/A
Contract	N/A
Network	N/A
Language	Solidity
Tax	N/A
Total Supply	N/A
Status	Not deployed

Current stats

Bonds per address	10

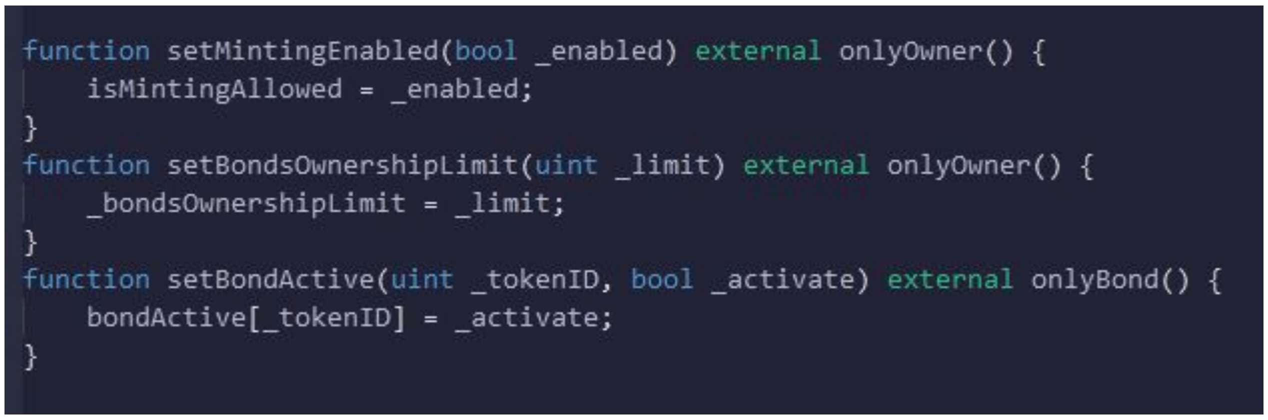
Security Threads

Owner can change the current minting contract address at any time.
⚠️Minting contract can mint new bonds.
⚠️Minting contract can burn existing bonds.



```
function setApprovedBond(address _bond, bool _enabled) external onlyOwner() {
    approvedBonds[_bond] = _enabled;
}
function mint(address _recipient, bool _activate) external onlyBond() returns (uint256) {
    require(isMintingAllowed, "Minting is disabled");
    require(this.balanceOf(msg.sender) < bondsOwnershipLimit(), "Bonds ownership limit reached");
    _currentTokenID = _currentTokenID + 1;
    bondActive[_currentTokenID] = _activate;
    bondMinter[_currentTokenID] = msg.sender;
    return _mintTo(_recipient, _currentTokenID);
}
function burn(uint _tokenID) external onlyBond() {
    _burn(_tokenID);
}
```

Owner can enable/disable minting.
Owner can set maximum bonds allowed per address.
Owner can enable/disable bond's active status. Active bonds cannot be transferred.



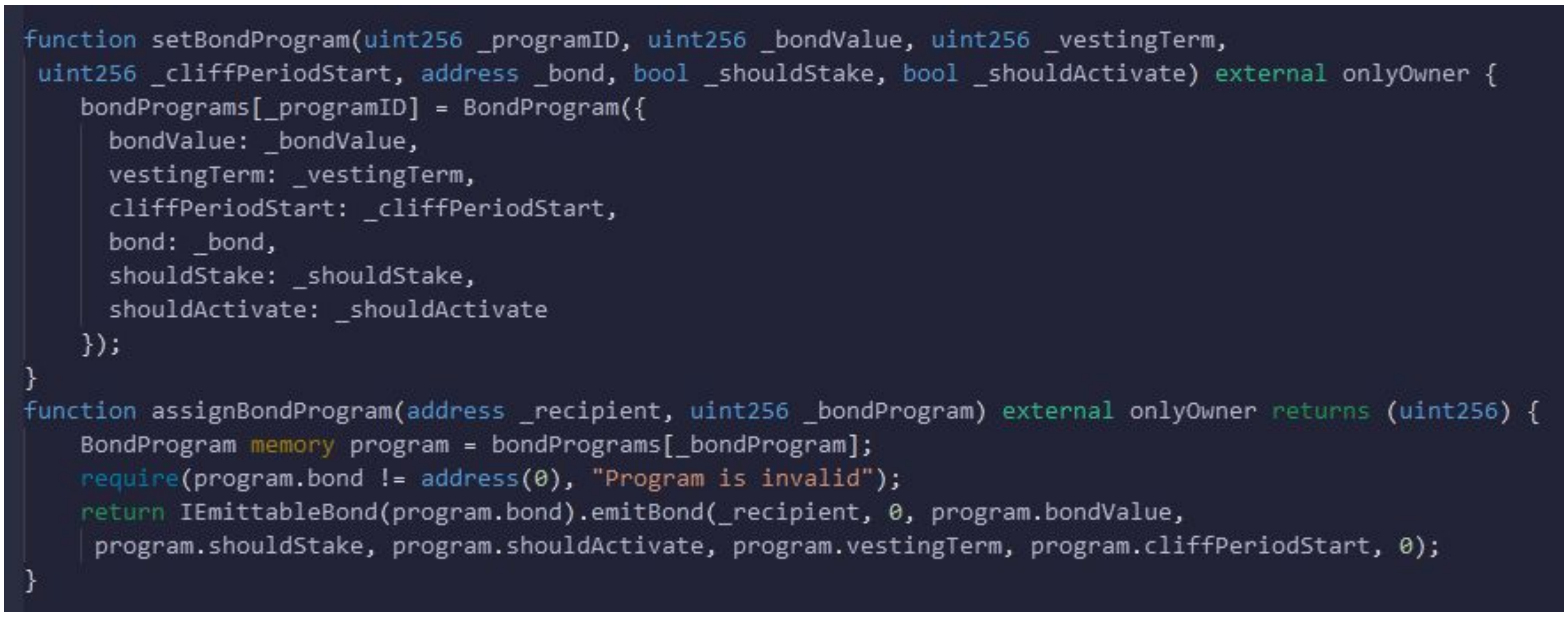
```
function setMintingEnabled(bool _enabled) external onlyOwner() {
    isMintingAllowed = _enabled;
}
function setBondsOwnershipLimit(uint _limit) external onlyOwner() {
    _bondsOwnershipLimit = _limit;
}
function setBondActive(uint _tokenID, bool _activate) external onlyBond() {
    bondActive[_tokenID] = _activate;
}
```

Smart Contract Summary

File name	**bondEmitter.sol**
Contract Name	N/A
Ticker	N/A
Contract	N/A
Network	N/A
Language	Solidity
Tax	N/A
Total Supply	N/A
Status	Not deployed

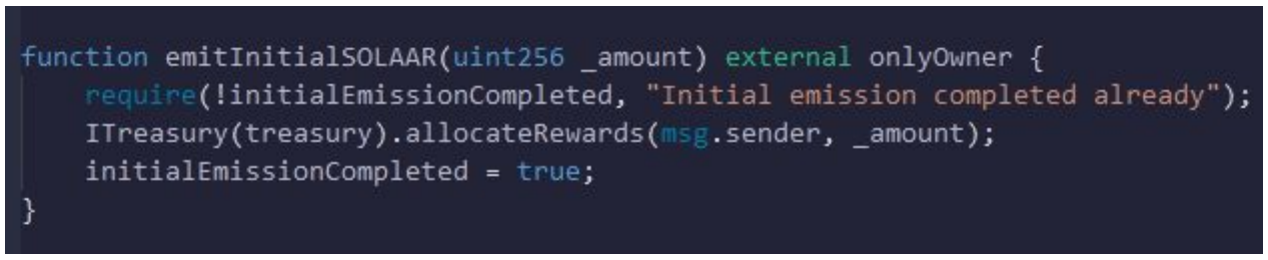
Security Threads

Owner can set and assign bond programs.



```
function setBondProgram(uint256 _programID, uint256 _bondValue, uint256 _vestingTerm,
 uint256 _cliffPeriodStart, address _bond, bool _shouldStake, bool _shouldActivate) external onlyOwner {
    bondPrograms[_programID] = BondProgram({
      bondValue: _bondValue,
      vestingTerm: _vestingTerm,
      cliffPeriodStart: _cliffPeriodStart,
      bond: _bond,
      shouldStake: _shouldStake,
      shouldActivate: _shouldActivate
    });
}
function assignBondProgram(address _recipient, uint256 _bondProgram) external onlyOwner returns (uint256) {
    BondProgram memory program = bondPrograms[_bondProgram];
    require(program.bond != address(0), "Program is invalid");
    return IEmittableBond(program.bond).emitBond(_recipient, 0, program.bondValue,
     program.shouldStake, program.shouldActivate, program.vestingTerm, program.cliffPeriodStart, 0);
}
```

Owner can mint new tokens once via the Treasury contract.



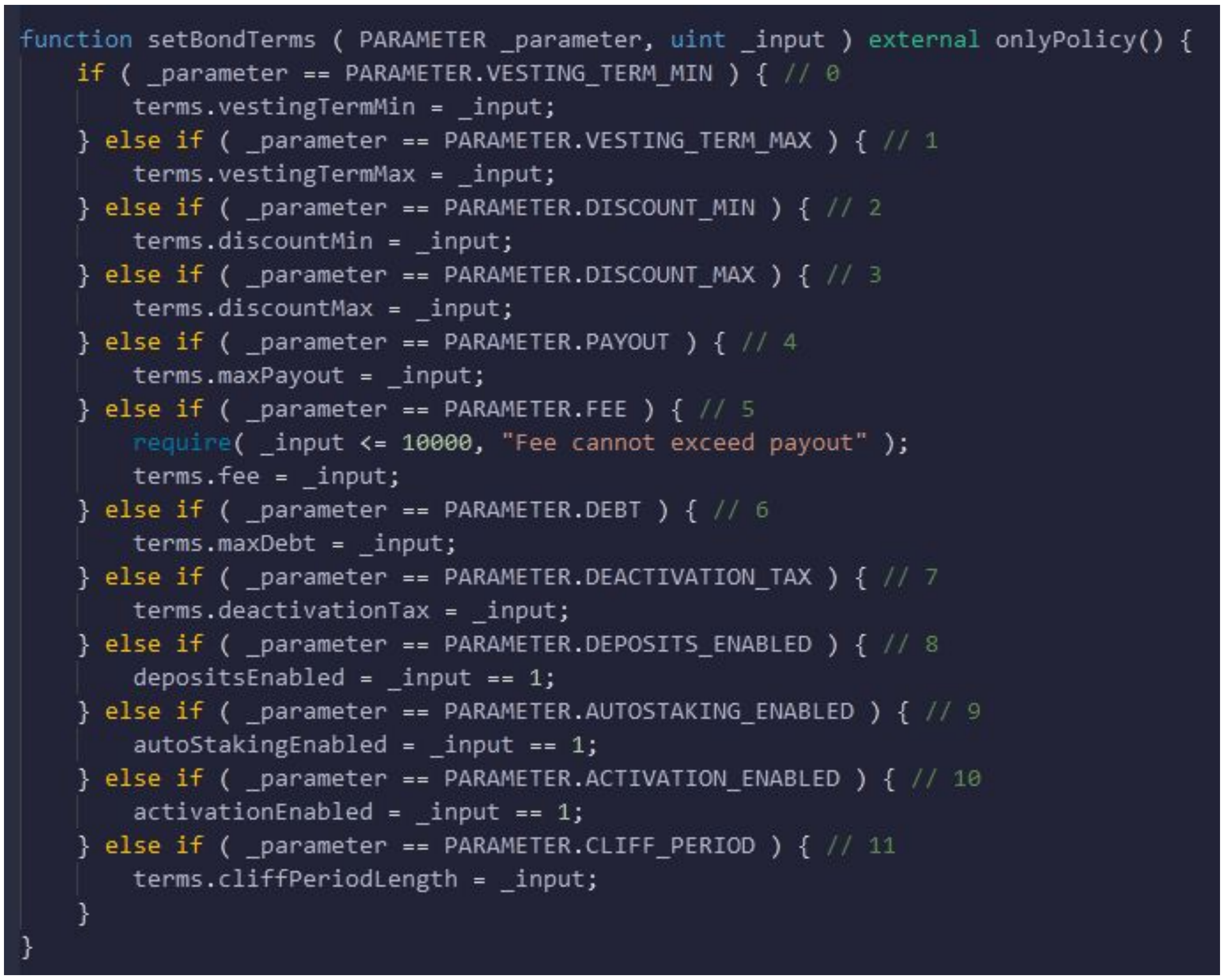
```
function emitInitialSOLAAR(uint256 _amount) external onlyOwner {
    require(!initialEmissionCompleted, "Initial emission completed already");
    ITreasury(treasury).allocateRewards(msg.sender, _amount);
    initialEmissionCompleted = true;
}
```

Smart Contract Summary

File name	**bond.sol**
Contract Name	N/A
Ticker	N/A
Contract	N/A
Network	N/A
Language	Solidity
Tax	N/A
Total Supply	N/A
Status	Not deployed

Security Threads

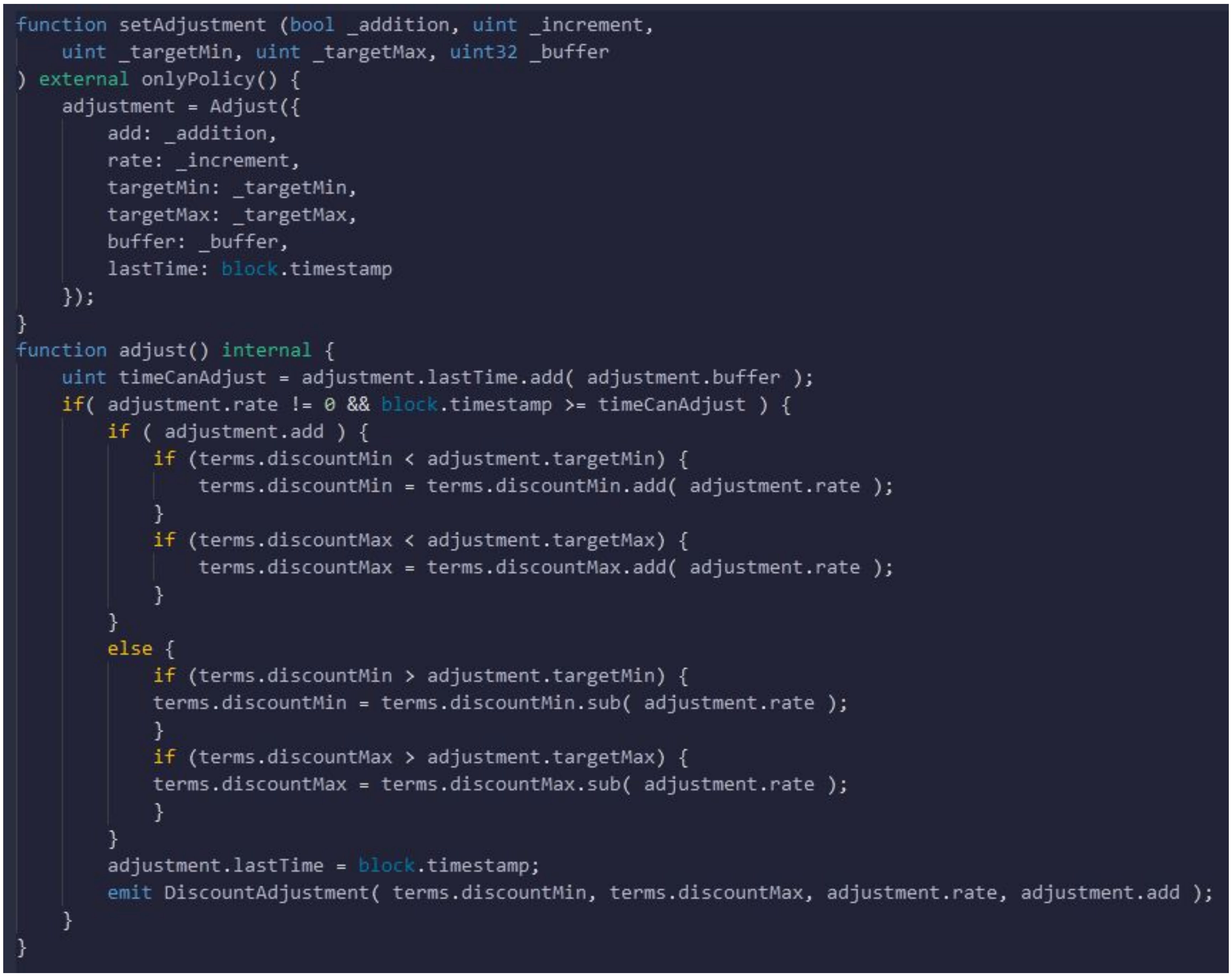
Owner can change bond parameters such as:
Vesting terms, discount, payout, fee, deactivation tax, cliff period.
⚠️ Fees can be set up to 100%.



```
function setBondTerms ( PARAMETER _parameter, uint _input ) external onlyPolicy() {
    if ( _parameter == PARAMETER.VESTING_TERM_MIN ) { // 0
        terms.vestingTermMin = _input;
    } else if ( _parameter == PARAMETER.VESTING_TERM_MAX ) { // 1
        terms.vestingTermMax = _input;
    } else if ( _parameter == PARAMETER.DISCOUNT_MIN ) { // 2
        terms.discountMin = _input;
    } else if ( _parameter == PARAMETER.DISCOUNT_MAX ) { // 3
        terms.discountMax = _input;
    } else if ( _parameter == PARAMETER.PAYOUT ) { // 4
        terms.maxPayout = _input;
    } else if ( _parameter == PARAMETER.FEE ) { // 5
        require( _input <= 10000, "Fee cannot exceed payout" );
        terms.fee = _input;
    } else if ( _parameter == PARAMETER.DEBT ) { // 6
        terms.maxDebt = _input;
    } else if ( _parameter == PARAMETER.DEACTIVATION_TAX ) { // 7
        terms.deactivationTax = _input;
    } else if ( _parameter == PARAMETER.DEPOSITS_ENABLED ) { // 8
        depositsEnabled = _input == 1;
    } else if ( _parameter == PARAMETER.AUTOSTAKING_ENABLED ) { // 9
        autoStakingEnabled = _input == 1;
    } else if ( _parameter == PARAMETER.ACTIVATION_ENABLED ) { // 10
        activationEnabled = _input == 1;
    } else if ( _parameter == PARAMETER.CLIFF_PERIOD ) { // 11
        terms.cliffPeriodLength = _input;
    }
}
```

Security Threads

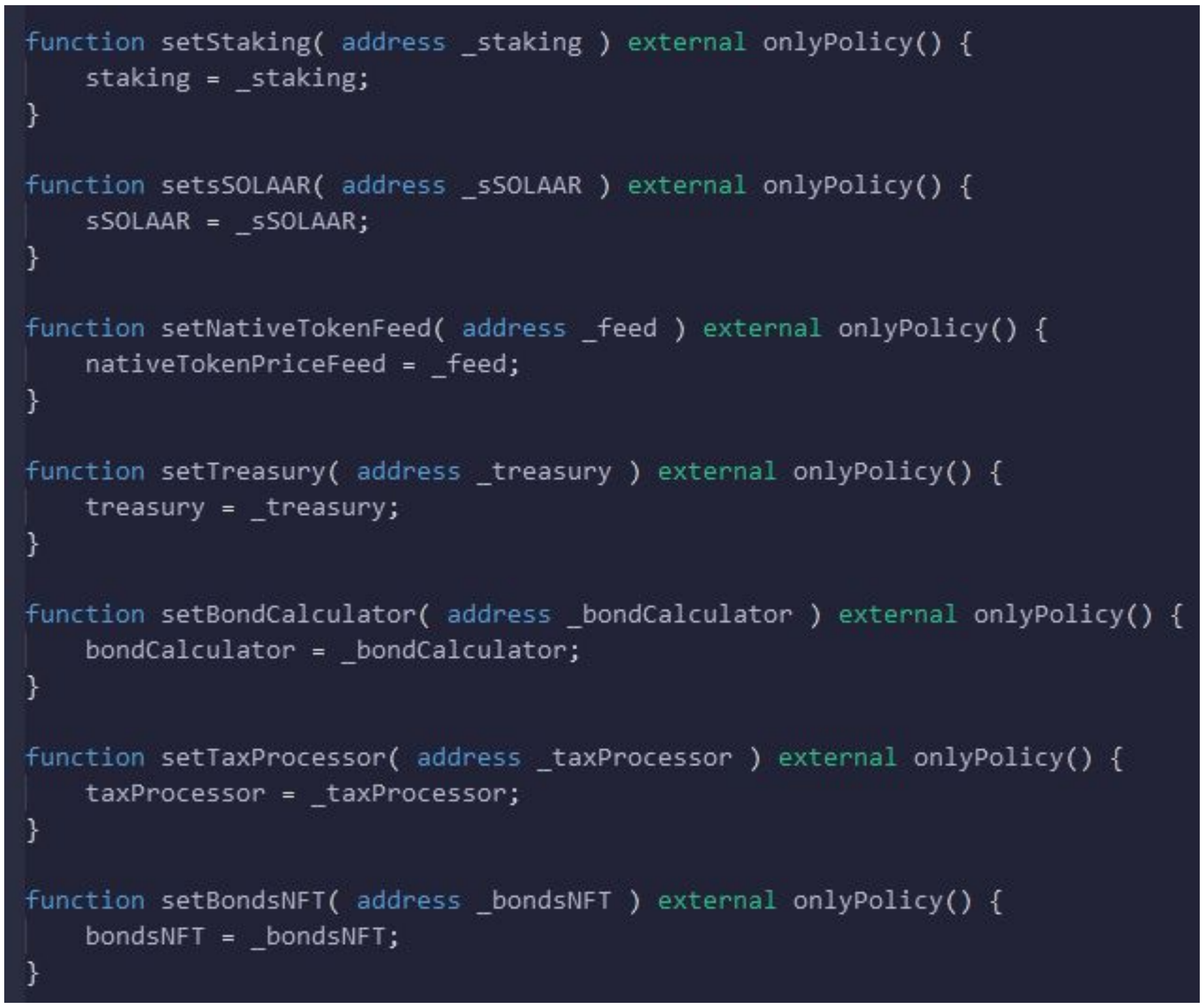
Owner can set bond adjustments.



```
function setAdjustment (bool _addition, uint _increment,
    uint _targetMin, uint _targetMax, uint32 _buffer
) external onlyPolicy() {
    adjustment = Adjust({
        add: _addition,
        rate: _increment,
        targetMin: _targetMin,
        targetMax: _targetMax,
        buffer: _buffer,
        lastTime: block.timestamp
    });
}
function adjust() internal {
    uint timeCanAdjust = adjustment.lastTime.add( adjustment.buffer );
    if( adjustment.rate != 0 && block.timestamp >= timeCanAdjust ) {
        if ( adjustment.add ) {
            if (terms.discountMin < adjustment.targetMin) {
                terms.discountMin = terms.discountMin.add( adjustment.rate );
            }
            if (terms.discountMax < adjustment.targetMax) {
                terms.discountMax = terms.discountMax.add( adjustment.rate );
            }
        }
        else {
            if (terms.discountMin > adjustment.targetMin) {
            terms.discountMin = terms.discountMin.sub( adjustment.rate );
            }
            if (terms.discountMax > adjustment.targetMax) {
            terms.discountMax = terms.discountMax.sub( adjustment.rate );
            }
        }
        adjustment.lastTime = block.timestamp;
        emit DiscountAdjustment( terms.discountMin, terms.discountMax, adjustment.rate, adjustment.add );
    }
}
```

Security Threads

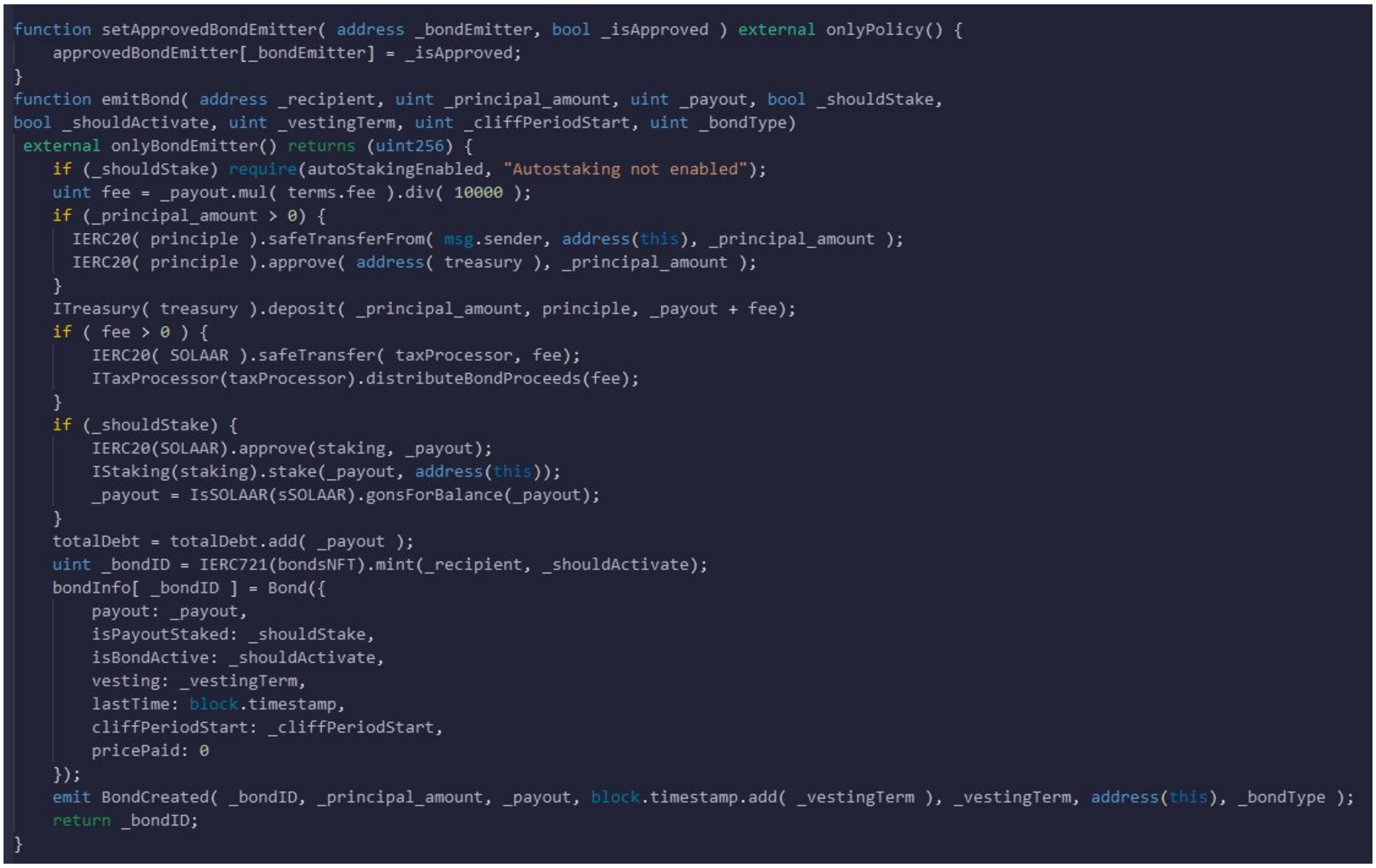
⚠️**Owner can change contract addresses as follows:**
Staking contract, sSolar token contract, Token price feed contract, Treasury minting contract, BondCalculator contract, TaxProcessor contract, BondsNft contract.



```
function setStaking( address _staking ) external onlyPolicy() {
    staking = _staking;
}

function setsSOLAAR( address _sSOLAAR ) external onlyPolicy() {
    sSOLAAR = _sSOLAAR;
}

function setNativeTokenFeed( address _feed ) external onlyPolicy() {
    nativeTokenPriceFeed = _feed;
}

function setTreasury( address _treasury ) external onlyPolicy() {
    treasury = _treasury;
}

function setBondCalculator( address _bondCalculator ) external onlyPolicy() {
    bondCalculator = _bondCalculator;
}

function setTaxProcessor( address _taxProcessor ) external onlyPolicy() {
    taxProcessor = _taxProcessor;
}

function setBondsNFT( address _bondsNFT ) external onlyPolicy() {
    bondsNFT = _bondsNFT;
}
```

Security Threads

Owner can change bond emitter authorized address.
Bond emitter can mint new bonds via bond_nft contract.



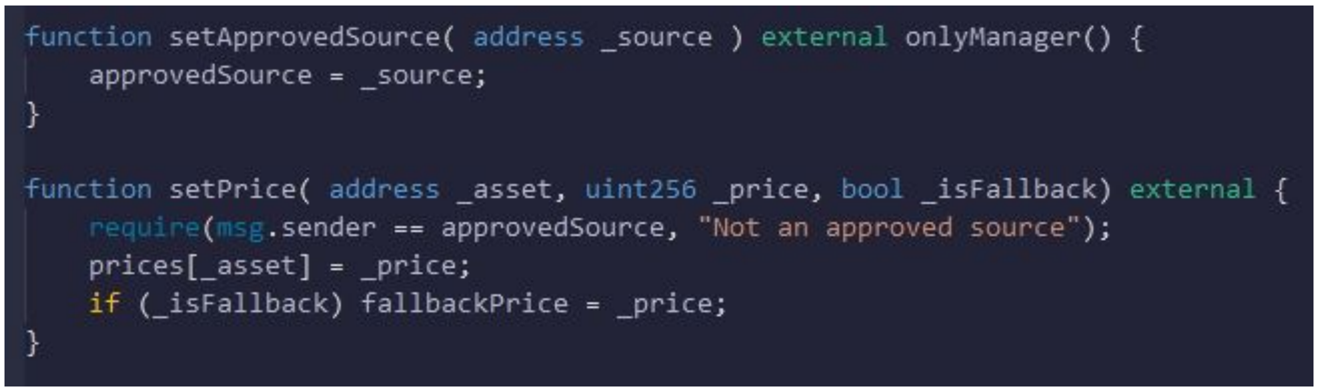
```
function setApprovedBondEmitter( address _bondEmitter, bool _isApproved ) external onlyPolicy() {
    approvedBondEmitter[_bondEmitter] = _isApproved;
}
function emitBond( address _recipient, uint _principal_amount, uint _payout, bool _shouldStake,
bool _shouldActivate, uint _vestingTerm, uint _cliffPeriodStart, uint _bondType)
 external onlyBondEmitter() returns (uint256) {
    if (_shouldStake) require(autoStakingEnabled, "Autostaking not enabled");
    uint fee = _payout.mul( terms.fee ).div( 10000 );
    if (_principal_amount > 0) {
      IERC20( principle ).safeTransferFrom( msg.sender, address(this), _principal_amount );
      IERC20( principle ).approve( address( treasury ), _principal_amount );
    }
    ITreasury( treasury ).deposit( _principal_amount, principle, _payout + fee);
    if ( fee > 0 ) {
        IERC20( SOLAAR ).safeTransfer( taxProcessor, fee);
        ITaxProcessor(taxProcessor).distributeBondProceeds(fee);
    }
    if (_shouldStake) {
        IERC20(SOLAAR).approve(staking, _payout);
        IStaking(staking).stake(_payout, address(this));
        _payout = IsSOLAAR(sSOLAAR).gonsForBalance(_payout);
    }
    totalDebt = totalDebt.add( _payout );
    uint _bondID = IERC721(bondsNFT).mint(_recipient, _shouldActivate);
    bondInfo[ _bondID ] = Bond({
        payout: _payout,
        isPayoutStaked: _shouldStake,
        isBondActive: _shouldActivate,
        vesting: _vestingTerm,
        lastTime: block.timestamp,
        cliffPeriodStart: _cliffPeriodStart,
        pricePaid: 0
    });
    emit BondCreated( _bondID, _principal_amount, _payout, block.timestamp.add( _vestingTerm ), _vestingTerm, address(this), _bondType );
    return _bondID;
}
```

Smart Contract Summary

File name	**asset_oracle_solaar.sol**
Contract Name	N/A
Ticker	N/A
Contract	N/A
Network	N/A
Language	Solidity
Tax	N/A
Total Supply	N/A
Status	Not deployed

Security Threads

Owner can change approved source address.
Approved source address can set prices for each asset.



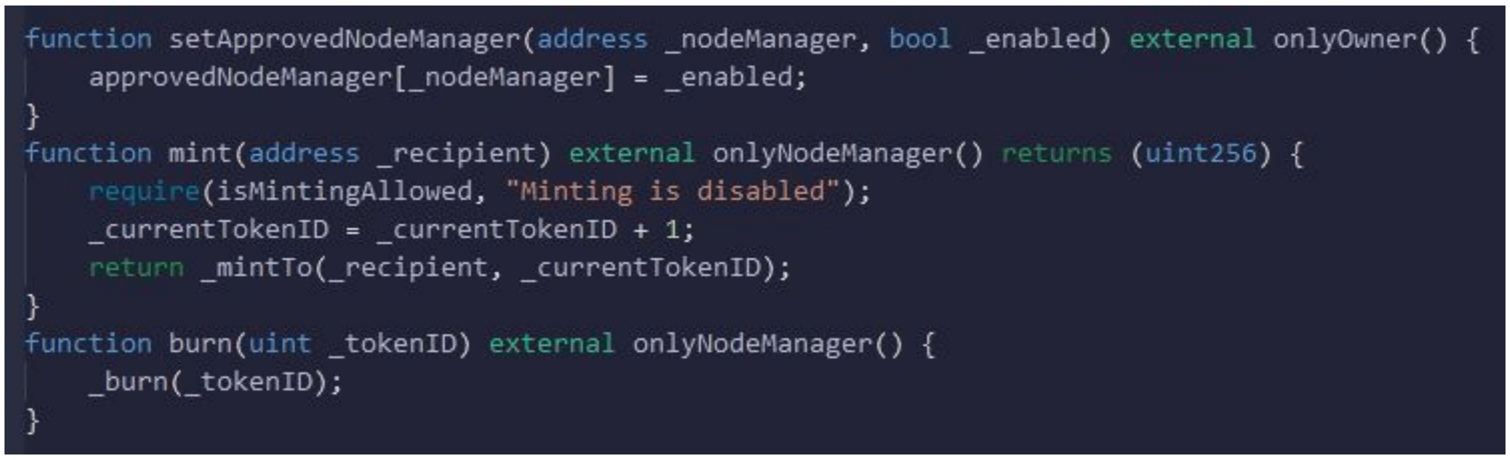
```
function setApprovedSource( address _source ) external onlyManager() {
    approvedSource = _source;
}

function setPrice( address _asset, uint256 _price, bool _isFallback) external {
    require(msg.sender == approvedSource, "Not an approved source");
    prices[_asset] = _price;
    if (_isFallback) fallbackPrice = _price;
}
```

Smart Contract Summary

File name	**node_nft.sol**
Contract Name	N/A
Ticker	N/A
Contract	N/A
Network	N/A
Language	Solidity
Tax	N/A
Total Supply	N/A
Status	Not deployed

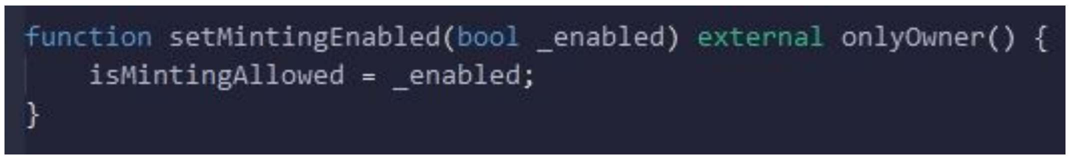
Security Threads

Owner can set node manager address.
⚠️Node manager can mint new nodes.
⚠️Node manager can burn existing nodes.



```
function setApprovedNodeManager(address _nodeManager, bool _enabled) external onlyOwner() {
    approvedNodeManager[_nodeManager] = _enabled;
}
function mint(address _recipient) external onlyNodeManager() returns (uint256) {
    require(isMintingAllowed, "Minting is disabled");
    _currentTokenID = _currentTokenID + 1;
    return _mintTo(_recipient, _currentTokenID);
}
function burn(uint _tokenID) external onlyNodeManager() {
    _burn(_tokenID);
}
```

Owner can enable/disable minting.



```
function setMintingEnabled(bool _enabled) external onlyOwner() {
    isMintingAllowed = _enabled;
}
```

Smart Contract Summary

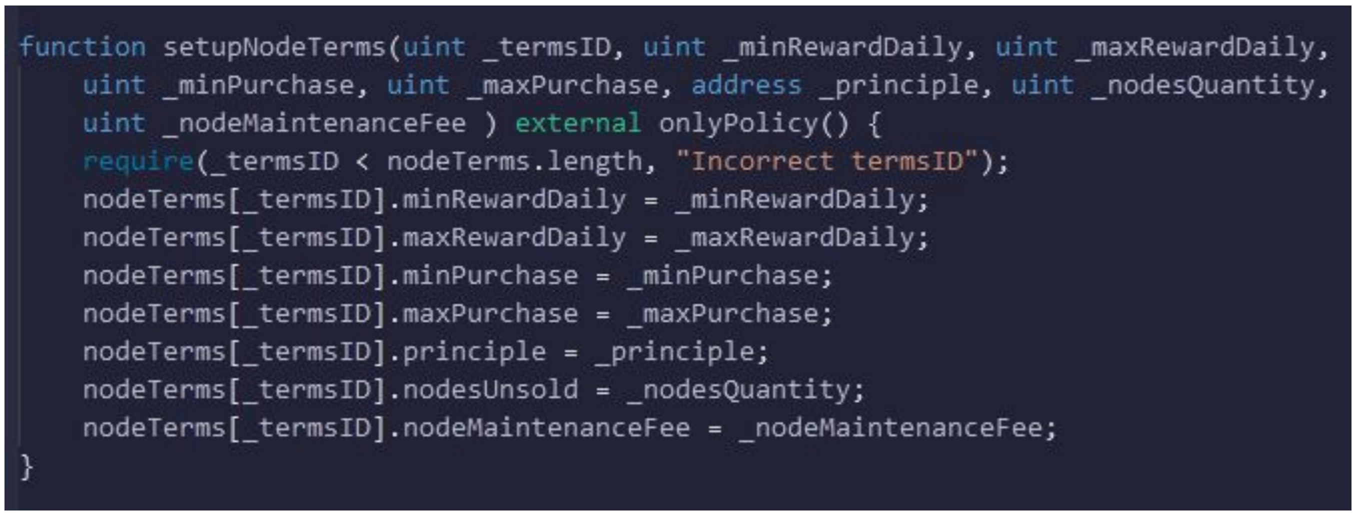
File name	**nodes.sol**
Contract Name	N/A
Ticker	N/A
Contract	N/A
Network	N/A
Language	Solidity
Tax	N/A
Total Supply	N/A
Status	Not deployed

Current stats

Min claim tax	0.5%
Max claim tax	20%
Linear period	90 days
Node life duration	365 days
Node maintenance frequency	90 days
Node maintenance termination	120 days - if maintenance fees are not paid, after this period node will be forfeited

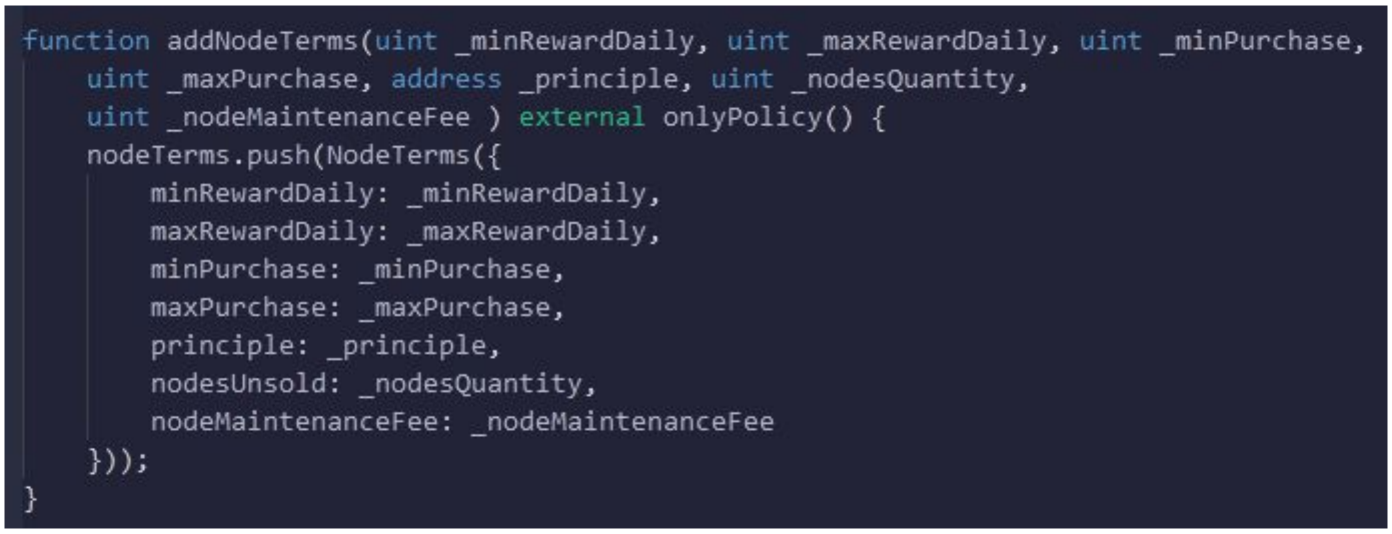
Security Threads

**Owner can set up node terms such as:
rewards paying, min/max node purchase, purchase currency, maintenance fee, max available nodes for purchase.**



```
function setupNodeTerms(uint _termsID, uint _minRewardDaily, uint _maxRewardDaily,
    uint _minPurchase, uint _maxPurchase, address _principle, uint _nodesQuantity,
    uint _nodeMaintenanceFee ) external onlyPolicy() {
    require(_termsID < nodeTerms.length, "Incorrect termsID");
    nodeTerms[_termsID].minRewardDaily = _minRewardDaily;
    nodeTerms[_termsID].maxRewardDaily = _maxRewardDaily;
    nodeTerms[_termsID].minPurchase = _minPurchase;
    nodeTerms[_termsID].maxPurchase = _maxPurchase;
    nodeTerms[_termsID].principle = _principle;
    nodeTerms[_termsID].nodesUnsold = _nodesQuantity;
    nodeTerms[_termsID].nodeMaintenanceFee = _nodeMaintenanceFee;
}
```

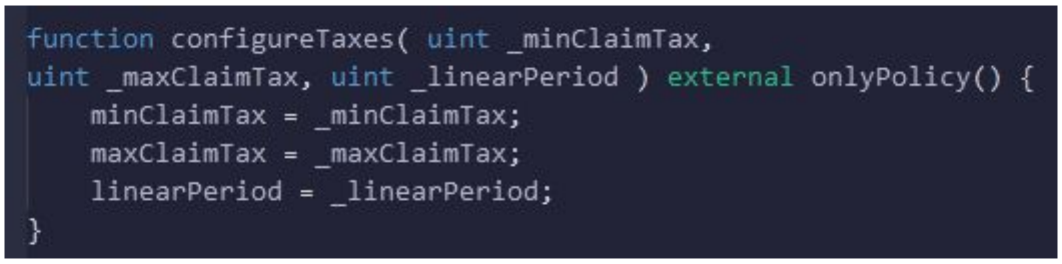
Owner can change node terms.



```
function addNodeTerms(uint _minRewardDaily, uint _maxRewardDaily, uint _minPurchase,
    uint _maxPurchase, address _principle, uint _nodesQuantity,
    uint _nodeMaintenanceFee ) external onlyPolicy() {
    nodeTerms.push(NodeTerms({
        minRewardDaily: _minRewardDaily,
        maxRewardDaily: _maxRewardDaily,
        minPurchase: _minPurchase,
        maxPurchase: _maxPurchase,
        principle: _principle,
        nodesUnsold: _nodesQuantity,
        nodeMaintenanceFee: _nodeMaintenanceFee
    }));
}
```

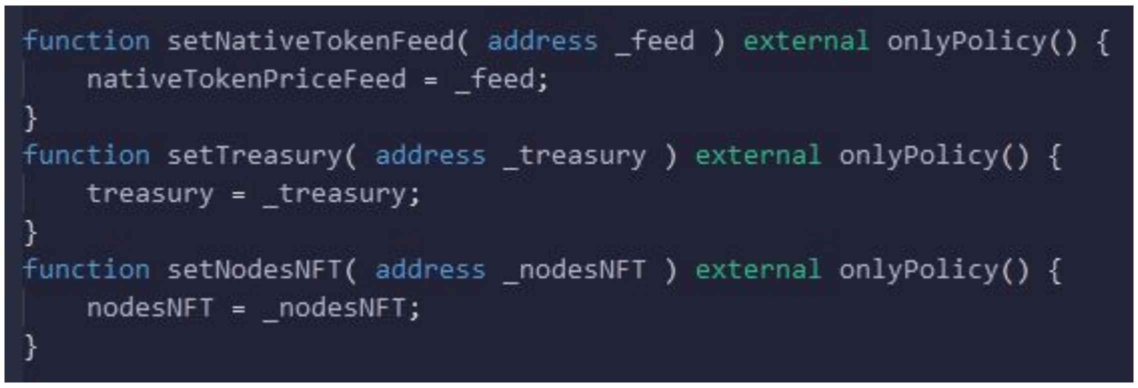
Security Threads

Owner can change node's claim taxes and periods.



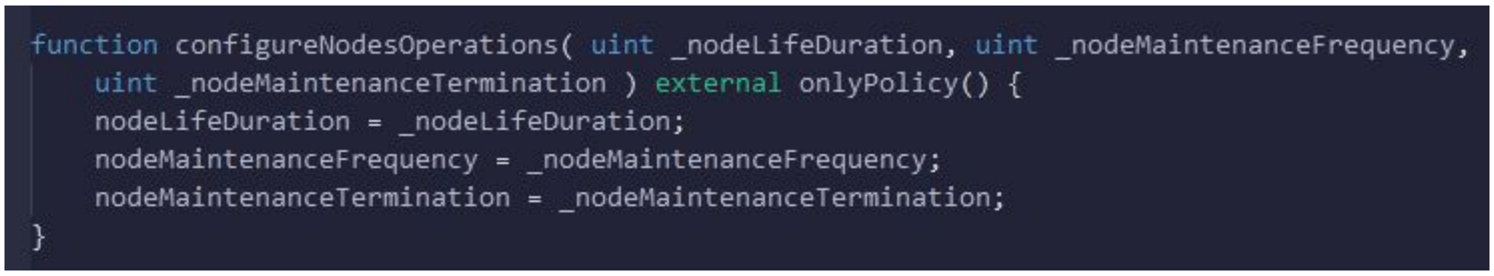
```
function configureTaxes( uint _minClaimTax,
uint _maxClaimTax, uint _linearPeriod ) external onlyPolicy() {
    minClaimTax = _minClaimTax;
    maxClaimTax = _maxClaimTax;
    linearPeriod = _linearPeriod;
}
```

Owner can change price feed contract, treasury contract, NFT contract.



```
function setNativeTokenFeed( address _feed ) external onlyPolicy() {
    nativeTokenPriceFeed = _feed;
}
function setTreasury( address _treasury ) external onlyPolicy() {
    treasury = _treasury;
}
function setNodesNFT( address _nodesNFT ) external onlyPolicy() {
    nodesNFT = _nodesNFT;
}
```

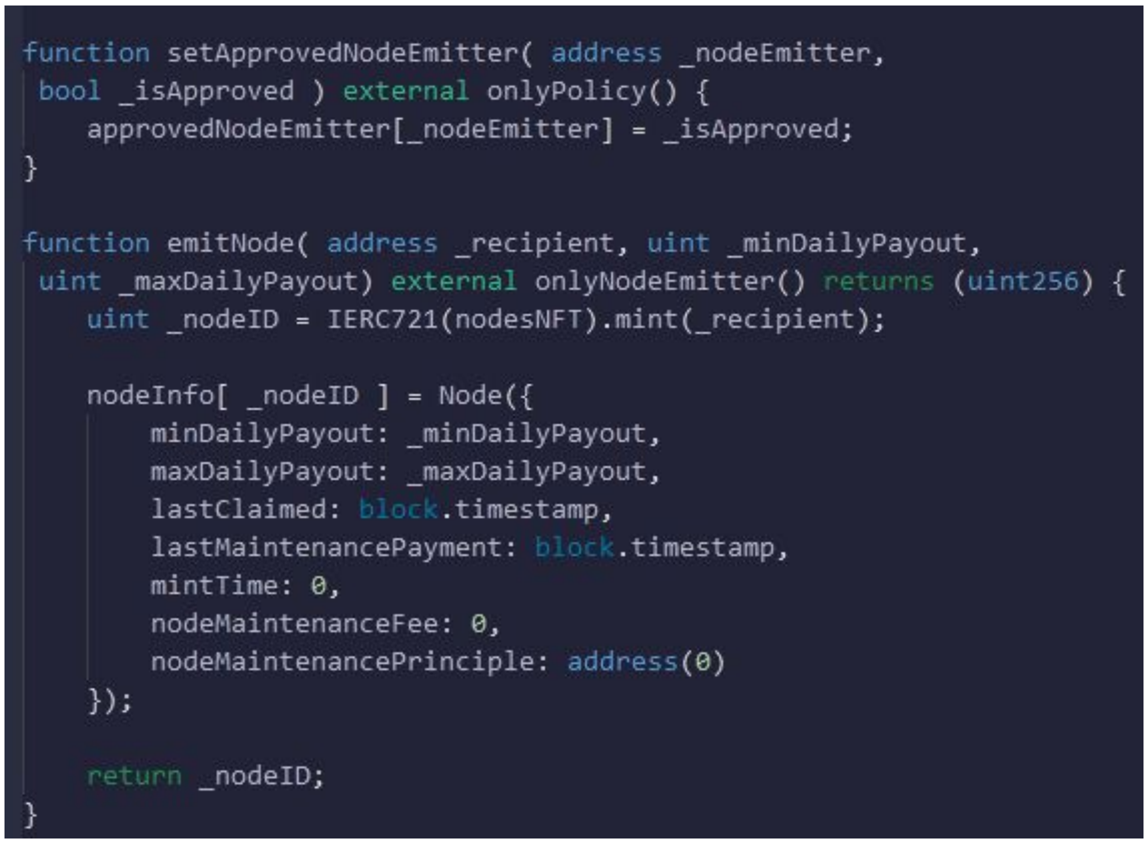
Owner can change node's reward production period, maintenance fees and forfeit time if maintenance fees are not paid.



```
function configureNodesOperations( uint _nodeLifeDuration, uint _nodeMaintenanceFrequency,
    uint _nodeMaintenanceTermination ) external onlyPolicy() {
    nodeLifeDuration = _nodeLifeDuration;
    nodeMaintenanceFrequency = _nodeMaintenanceFrequency;
    nodeMaintenanceTermination = _nodeMaintenanceTermination;
}
```

Security Threads

Owner can change Node emitter address.
Node emitter can mint new nodes.



```
function setApprovedNodeEmitter( address _nodeEmitter,
 bool _isApproved ) external onlyPolicy() {
    approvedNodeEmitter[_nodeEmitter] = _isApproved;
}

function emitNode( address _recipient, uint _minDailyPayout,
 uint _maxDailyPayout) external onlyNodeEmitter() returns (uint256) {
    uint _nodeID = IERC721(nodesNFT).mint(_recipient);

    nodeInfo[ _nodeID ] = Node({
        minDailyPayout: _minDailyPayout,
        maxDailyPayout: _maxDailyPayout,
        lastClaimed: block.timestamp,
        lastMaintenancePayment: block.timestamp,
        mintTime: 0,
        nodeMaintenanceFee: 0,
        nodeMaintenancePrinciple: address(0)
    });

    return _nodeID;
}
```

Smart Contract Summary

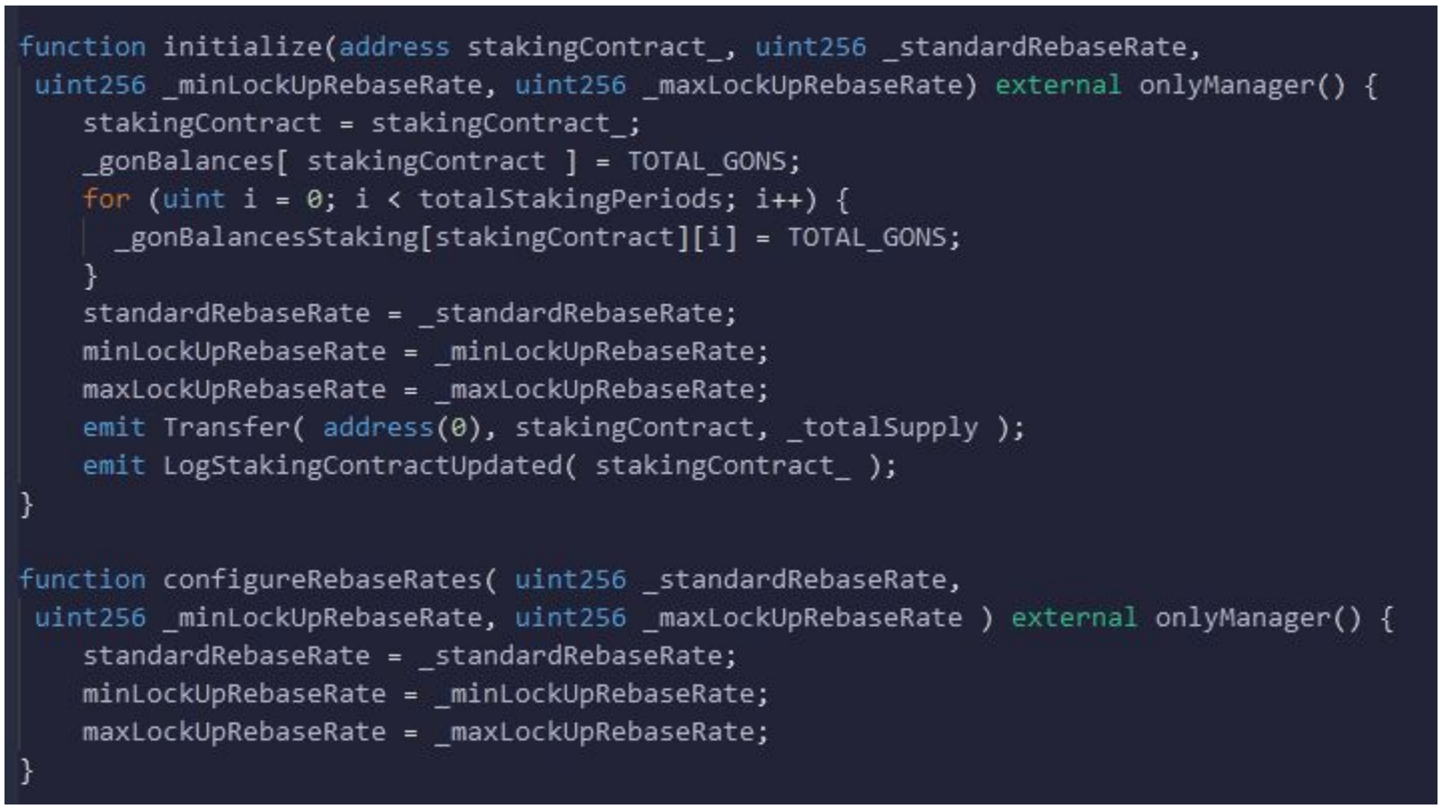
File name	**sSOLR.sol**
Contract Name	Assigned at deployment
Ticker	Assigned at deployment
Contract	N/A
Network	N/A
Language	Solidity
Tax	N/A
Initial Supply	5,000,000
Status	Not deployed

Current stats

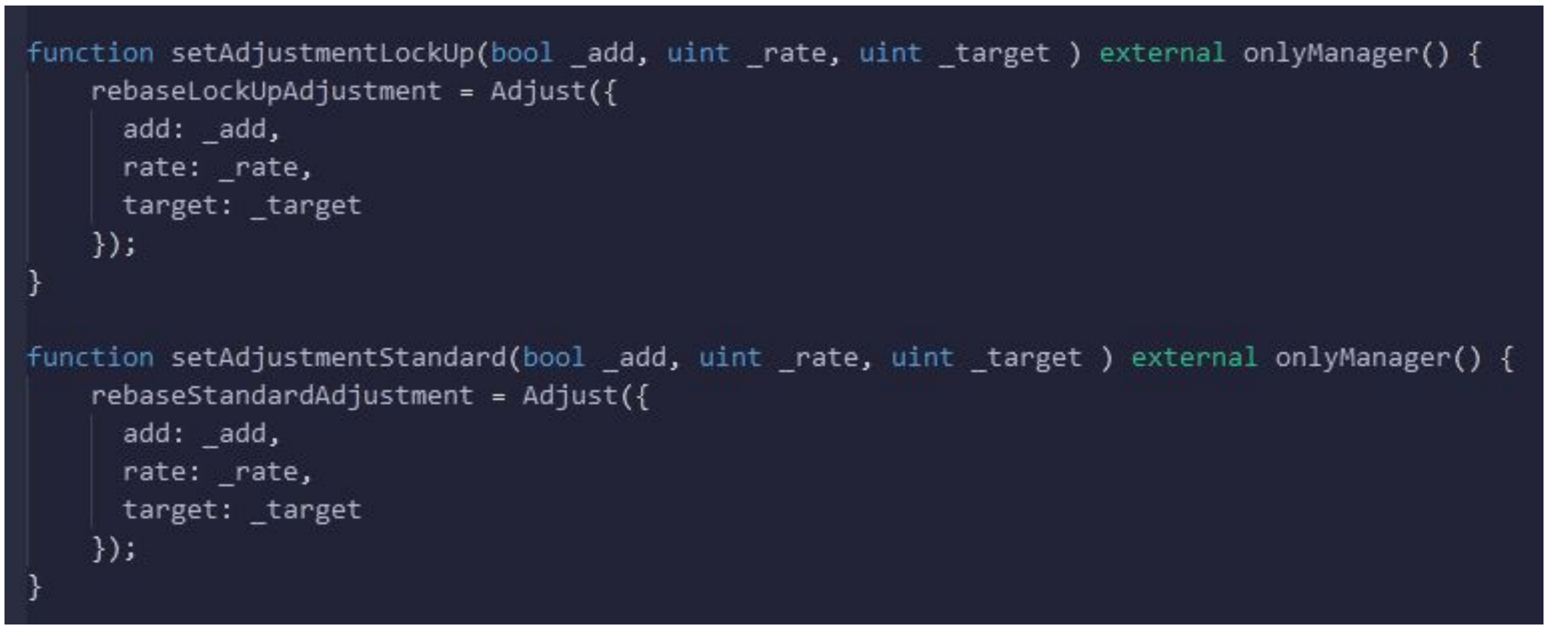
Standard rebase rate	Not initialized
Min lockup rebase rate	Not initialized
Max lockup rebase rate	Not initialized
Staking Contract	Not initialized

Security Threads

Owner can set staking contracts and change the rebase rates.

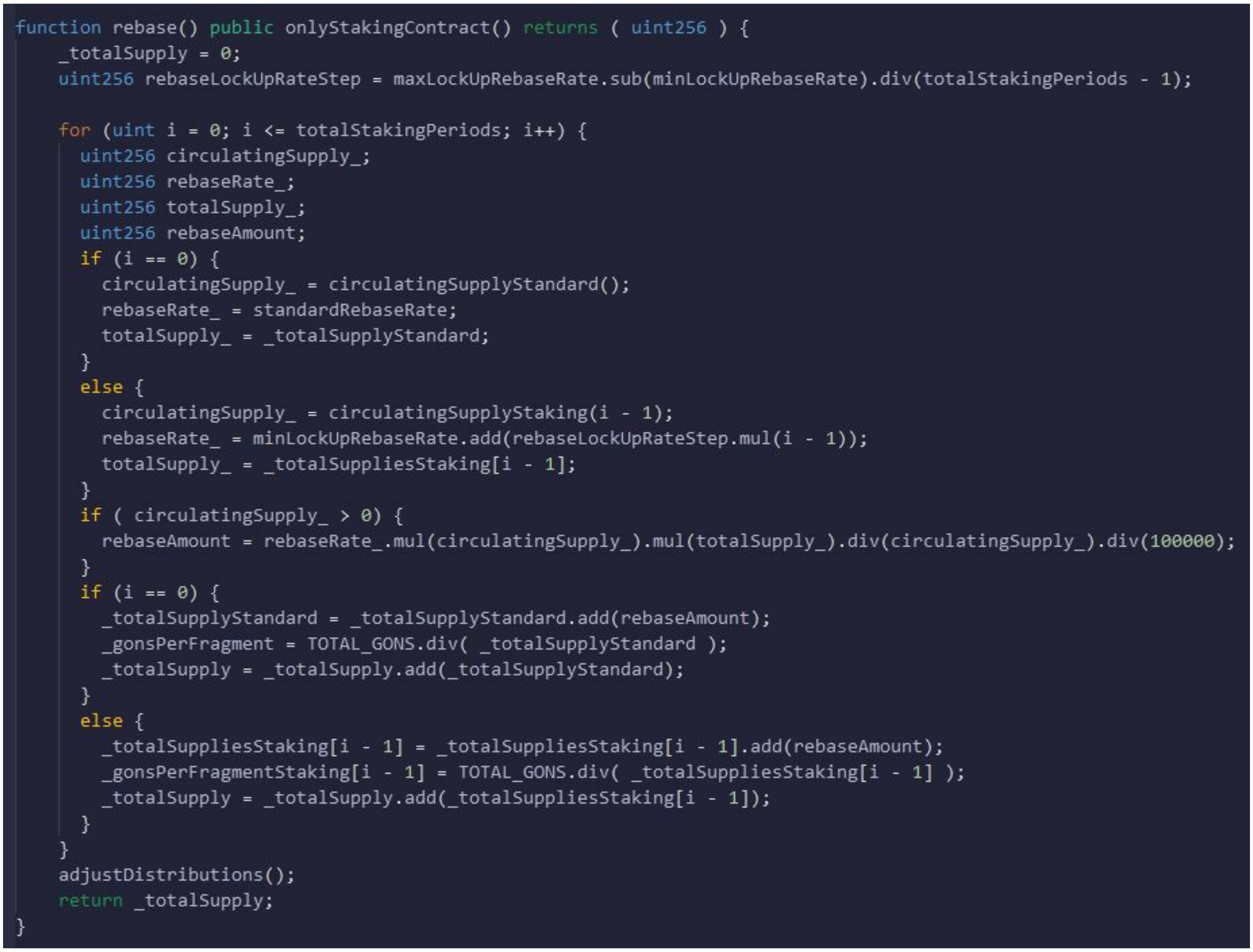

```
function initialize(address stakingContract_, uint256 _standardRebaseRate,
 uint256 _minLockUpRebaseRate, uint256 _maxLockUpRebaseRate) external onlyManager() {
    stakingContract = stakingContract_;
    _gonBalances[ stakingContract ] = TOTAL_GONS;
    for (uint i = 0; i < totalStakingPeriods; i++) {
      _gonBalancesStaking[stakingContract][i] = TOTAL_GONS;
    }
    standardRebaseRate = _standardRebaseRate;
    minLockUpRebaseRate = _minLockUpRebaseRate;
    maxLockUpRebaseRate = _maxLockUpRebaseRate;
    emit Transfer( address(0), stakingContract, _totalSupply );
    emit LogStakingContractUpdated( stakingContract_ );
}

function configureRebaseRates( uint256 _standardRebaseRate,
 uint256 _minLockUpRebaseRate, uint256 _maxLockUpRebaseRate ) external onlyManager() {
    standardRebaseRate = _standardRebaseRate;
    minLockUpRebaseRate = _minLockUpRebaseRate;
    maxLockUpRebaseRate = _maxLockUpRebaseRate;
}
```

Owner can change staking parameters.


```
function setAdjustmentLockUp(bool _add, uint _rate, uint _target ) external onlyManager() {
    rebaseLockUpAdjustment = Adjust({
      add: _add,
      rate: _rate,
      target: _target
    });
}

function setAdjustmentStandard(bool _add, uint _rate, uint _target ) external onlyManager() {
    rebaseStandardAdjustment = Adjust({
      add: _add,
      rate: _rate,
      target: _target
    });
}
```

Security Threads

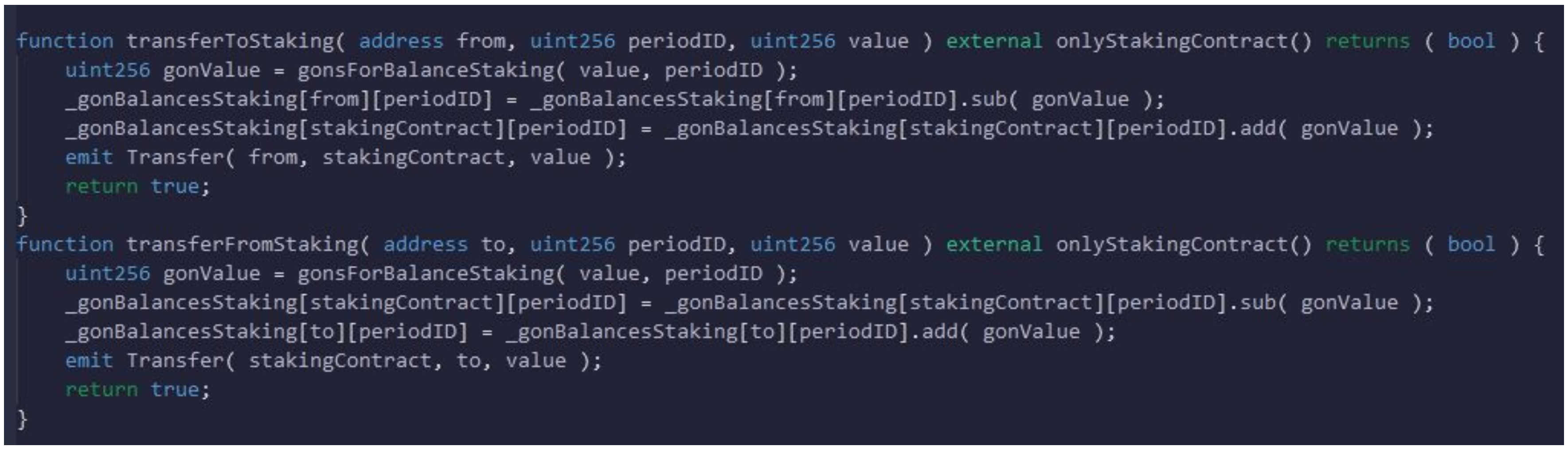
Staking contract can initiate token rebase.



```
function rebase() public onlyStakingContract() returns ( uint256 ) {
    _totalSupply = 0;
    uint256 rebaseLockUpRateStep = maxLockUpRebaseRate.sub(minLockUpRebaseRate).div(totalStakingPeriods - 1);

    for (uint i = 0; i <= totalStakingPeriods; i++) {
      uint256 circulatingSupply_;
      uint256 rebaseRate_;
      uint256 totalSupply_;
      uint256 rebaseAmount;
      if (i == 0) {
        circulatingSupply_ = circulatingSupplyStandard();
        rebaseRate_ = standardRebaseRate;
        totalSupply_ = _totalSupplyStandard;
      }
      else {
        circulatingSupply_ = circulatingSupplyStaking(i - 1);
        rebaseRate_ = minLockUpRebaseRate.add(rebaseLockUpRateStep.mul(i - 1));
        totalSupply_ = _totalSuppliesStaking[i - 1];
      }
      if ( circulatingSupply_ > 0) {
        rebaseAmount = rebaseRate_.mul(circulatingSupply_).mul(totalSupply_).div(circulatingSupply_).div(100000);
      }
      if (i == 0) {
        _totalSupplyStandard = _totalSupplyStandard.add(rebaseAmount);
        _gonsPerFragment = TOTAL_GONS.div( _totalSupplyStandard );
        _totalSupply = _totalSupply.add(_totalSupplyStandard);
      }
      else {
        _totalSuppliesStaking[i - 1] = _totalSuppliesStaking[i - 1].add(rebaseAmount);
        _gonsPerFragmentStaking[i - 1] = TOTAL_GONS.div( _totalSuppliesStaking[i - 1] );
        _totalSupply = _totalSupply.add(_totalSuppliesStaking[i - 1]);
      }
    }
    adjustDistributions();
    return _totalSupply;
}
```

Security Threads

Staking contract address can initiate transfers.



```
function transferToStaking( address from, uint256 periodID, uint256 value ) external onlyStakingContract() returns ( bool ) {
    uint256 gonValue = gonsForBalanceStaking( value, periodID );
    _gonBalancesStaking[from][periodID] = _gonBalancesStaking[from][periodID].sub( gonValue );
    _gonBalancesStaking[stakingContract][periodID] = _gonBalancesStaking[stakingContract][periodID].add( gonValue );
    emit Transfer( from, stakingContract, value );
    return true;
}
function transferFromStaking( address to, uint256 periodID, uint256 value ) external onlyStakingContract() returns ( bool ) {
    uint256 gonValue = gonsForBalanceStaking( value, periodID );
    _gonBalancesStaking[stakingContract][periodID] = _gonBalancesStaking[stakingContract][periodID].sub( gonValue );
    _gonBalancesStaking[to][periodID] = _gonBalancesStaking[to][periodID].add( gonValue );
    emit Transfer( stakingContract, to, value );
    return true;
}
```

Smart Contract Summary

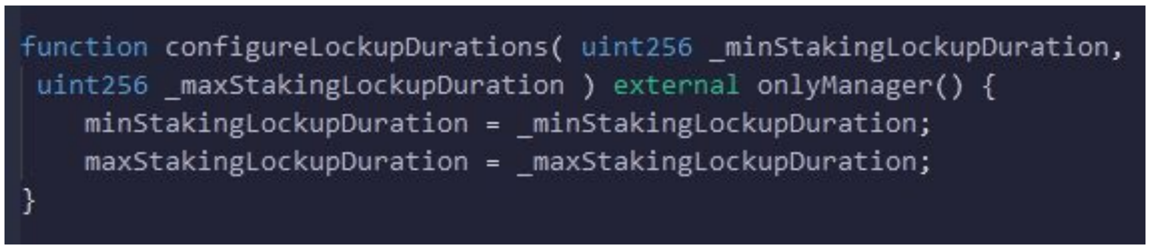
File name	**staking.sol**
Contract Name	N/A
Ticker	N/A
Contract	N/A
Network	N/A
Language	Solidity
Tax	N/A
Initial Supply	N/A
Status	Not deployed

Current stats

Treasury contract	Assigned at deployment
Min staking lockup duration	Assigned at deployment
Max staking lockup duration	Assigned at deployment
Randomizer contract	Assigned at deployment
Epoch length	Assigned at deployment

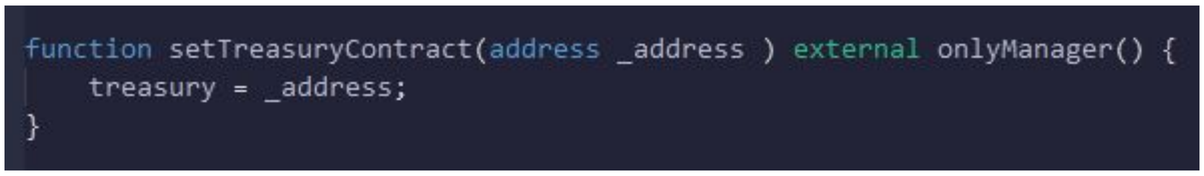
Security Threads

Owner can change staking periods.



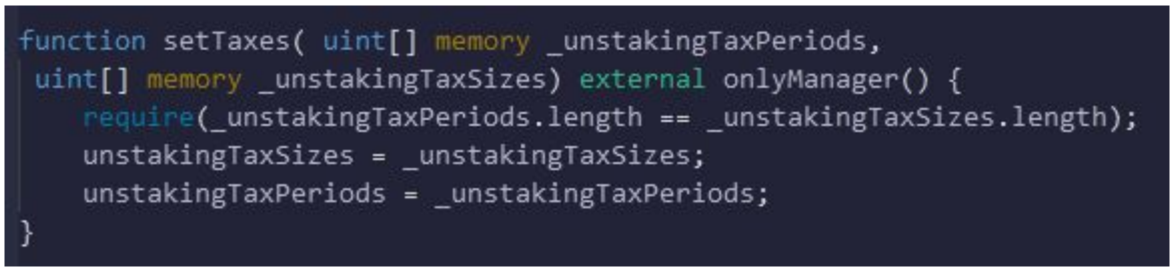
```
function configureLockupDurations( uint256 _minStakingLockupDuration,
 uint256 _maxStakingLockupDuration ) external onlyManager() {
    minStakingLockupDuration = _minStakingLockupDuration;
    maxStakingLockupDuration = _maxStakingLockupDuration;
}
```

Owner can change treasury contract.



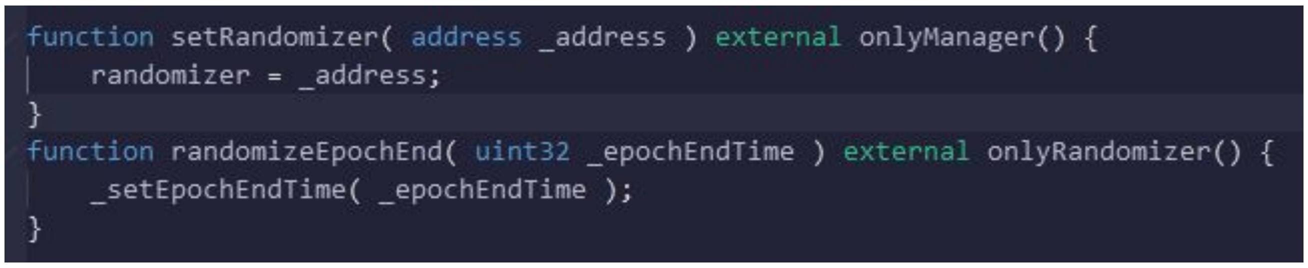
```
function setTreasuryContract(address _address ) external onlyManager() {
    treasury = _address;
}
```

Owner can change unstaking taxes.



```
function setTaxes( uint[] memory _unstakingTaxPeriods,
 uint[] memory _unstakingTaxSizes) external onlyManager() {
    require(_unstakingTaxPeriods.length == _unstakingTaxSizes.length);
    unstakingTaxSizes = _unstakingTaxSizes;
    unstakingTaxPeriods = _unstakingTaxPeriods;
}
```

Owner can change randomizer address.
Randomizer address can change rebase epoch time.



```
function setRandomizer( address _address ) external onlyManager() {
    randomizer = _address;
}
function randomizeEpochEnd( uint32 _epochEndTime ) external onlyRandomizer() {
    _setEpochEndTime( _epochEndTime );
}
```

Smart Contract Summary

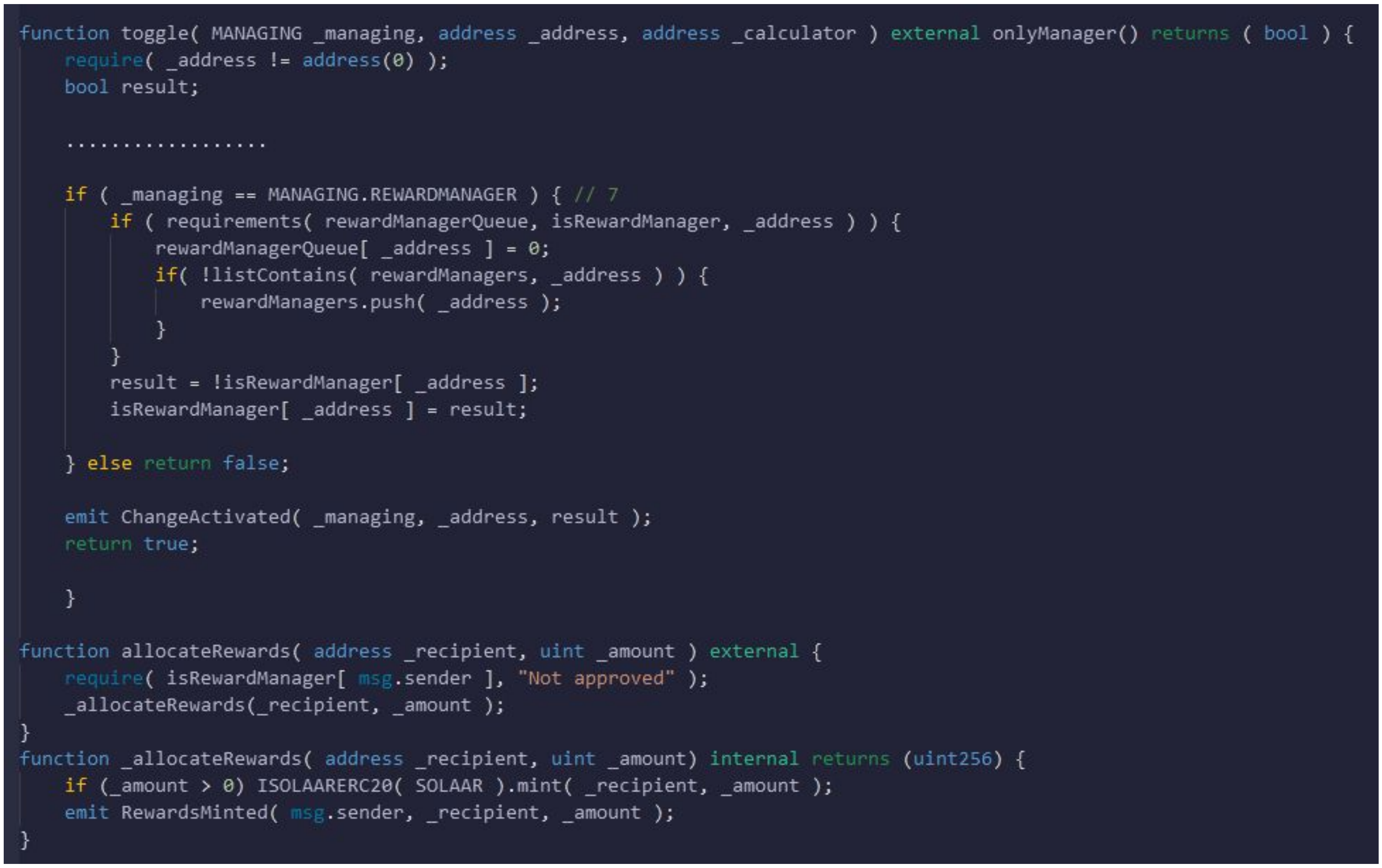
File name	**treasury.sol**
Contract Name	N/A
Ticker	N/A
Contract	N/A
Network	N/A
Language	Solidity
Tax	N/A
Initial Supply	N/A
Status	Not deployed

Current stats

Backing treasury allocation	20%

Security Threads

Owner can set rewards manager.
Rewards manager can mint new Solaar tokens.

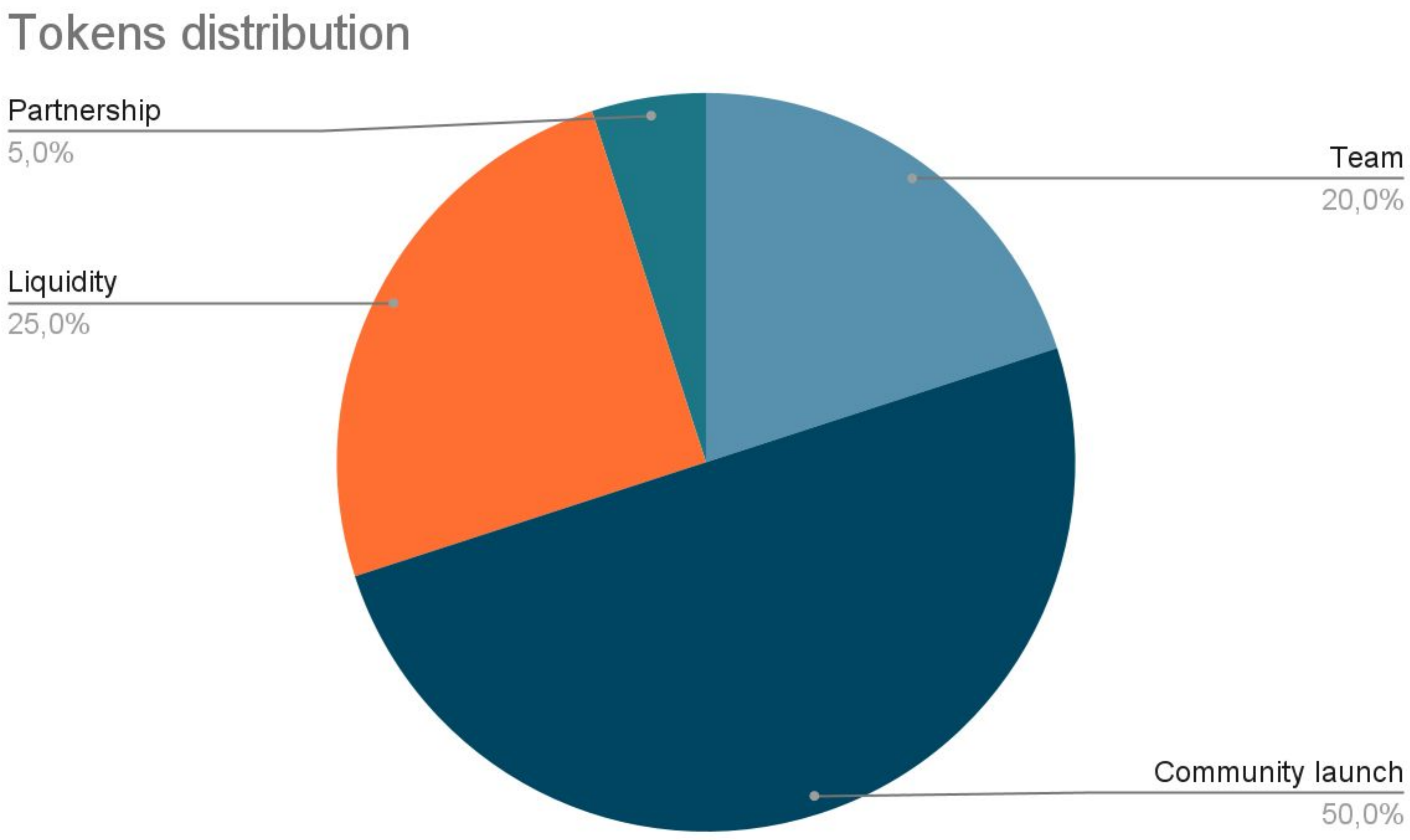


```
function toggle( MANAGING _managing, address _address, address _calculator ) external onlyManager() returns ( bool ) {
    require( _address != address(0) );
    bool result;

    .................

    if ( _managing == MANAGING.REWARDMANAGER ) { // 7
        if ( requirements( rewardManagerQueue, isRewardManager, _address ) ) {
            rewardManagerQueue[ _address ] = 0;
            if( !listContains( rewardManagers, _address ) ) {
                rewardManagers.push( _address );
            }
        }
        result = !isRewardManager[ _address ];
        isRewardManager[ _address ] = result;

    } else return false;

    emit ChangeActivated( _managing, _address, result );
    return true;

    }

function allocateRewards( address _recipient, uint _amount ) external {
    require( isRewardManager[ msg.sender ], "Not approved" );
    _allocateRewards(_recipient, _amount );
}
function _allocateRewards( address _recipient, uint _amount) internal returns (uint256) {
    if (_amount > 0) ISOLAARERC20( SOLAAR ).mint( _recipient, _amount );
    emit RewardsMinted( msg.sender, _recipient, _amount );
}
```

Tokenomics

According to their whitepaper

- 20% - Team
- 25% - Liquidity
- 5% - Partnerships
- 50% - Community launch NFT IBO



Contracts are not deployed yet.

Solaar
Project & Team Review

According to their whitepaper:
SOLAAR Protocol will be a decentralized reserve coin protocol on the Binance blockchain based on the $SOLAAR token.
Each $SOLAAR token will be backed by a basket of assets (e.g. UST/BNB/BTC) and LP tokens in the Solaar treasury, giving it an intrinsic value that provides a foundation of value. SOLAAR Protocol will also introduce economic and game-theoretic dynamics into the market through staking and bonds.

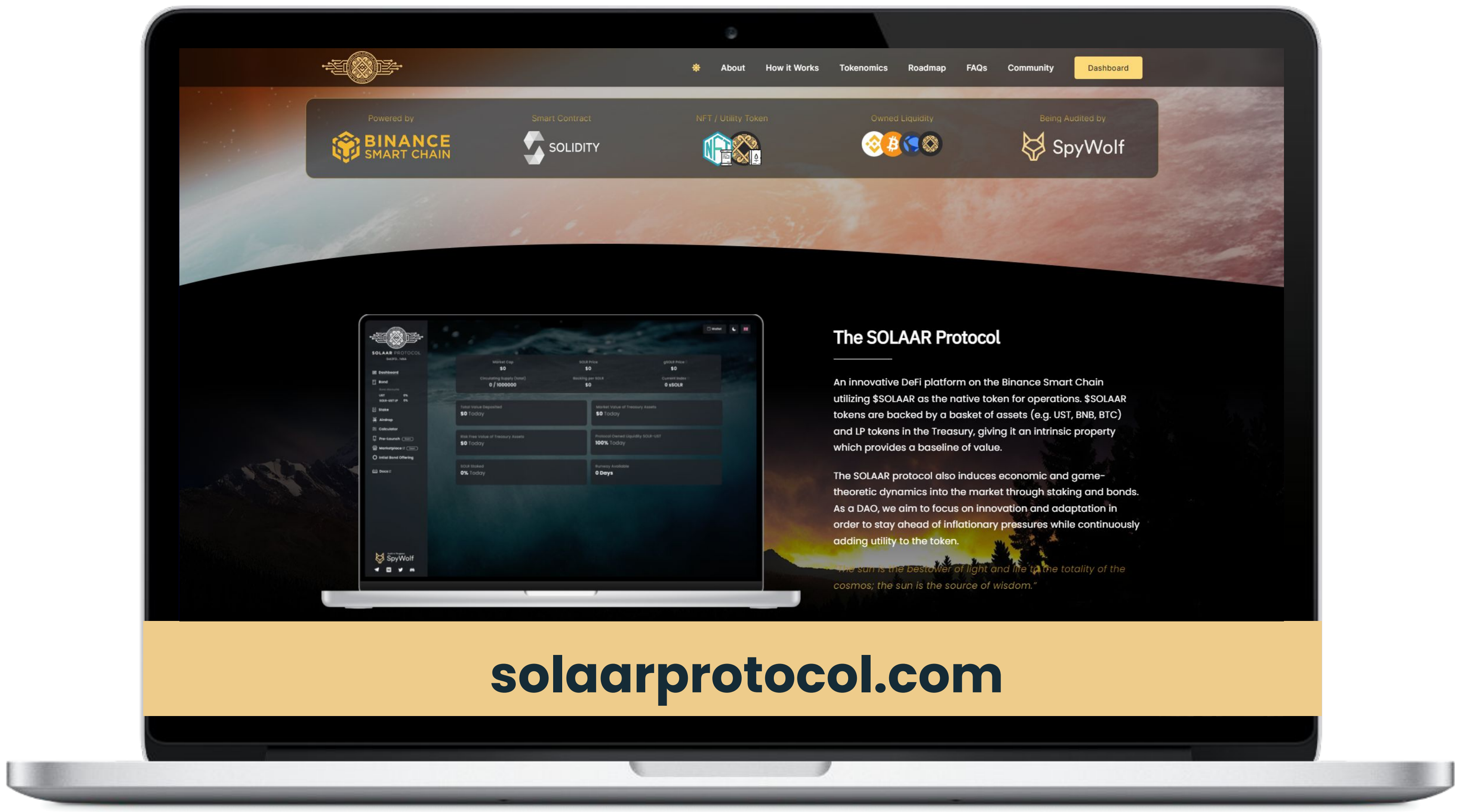
Team:

KYC not performed by SpyWolf

Website Analysis

URL: https://solaarprotocol.com/

- **Design:** Single page, pleasant design and color scheme.
- **Content:** Informative, no grammar mistakes.
- **Whitepaper:** Well written, explanatory, no grammar mistakes.
- **Roadmap:** Goals set for at least year a head with time frames.
- **Mobile-friendly?:** Yes
- **Technical:** SSL certificate present. General SEO check passed.



The SOLAAR Protocol

An innovative DeFi platform on the Binance Smart Chain utilizing $SOLAAR as the native token for operations. $SOLAAR tokens are backed by a basket of assets (e.g. UST, BNB, BTC) and LP tokens in the Treasury, giving it an intrinsic property which provides a baseline of value.

The SOLAAR protocol also induces economic and game-theoretic dynamics into the market through staking and bonds. As a DAO, we aim to focus on innovation and adaptation in order to stay ahead of inflationary pressures while continuously adding utility to the token.

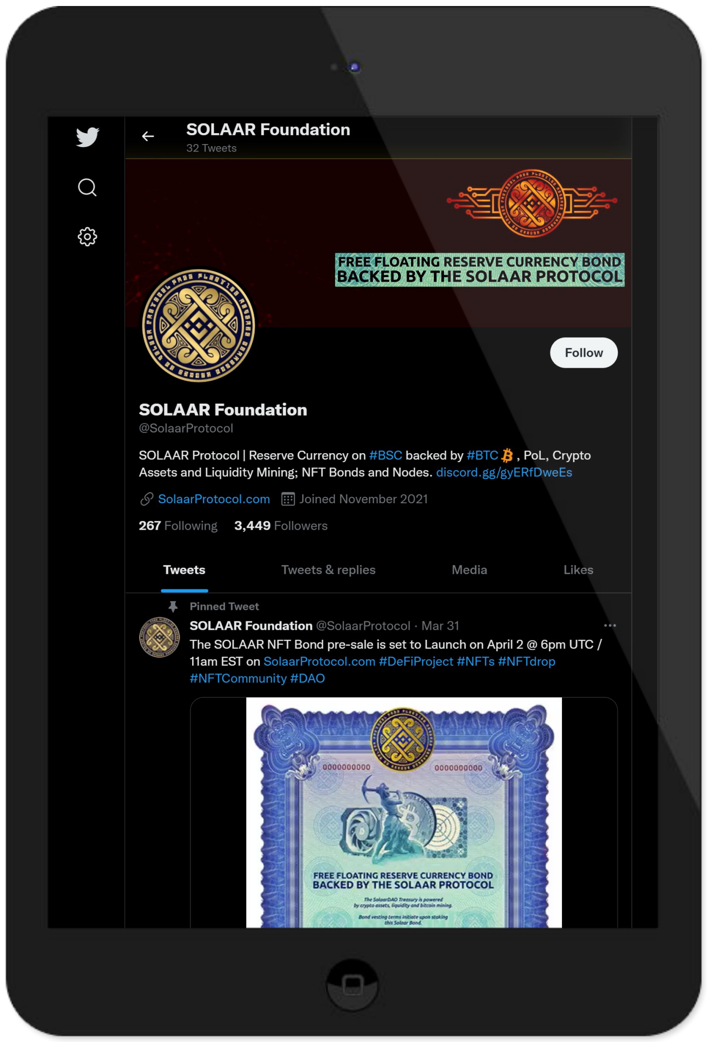
"The sun is the bestower of light and life to the totality of the cosmos; the sun is the source of wisdom."

Social Media & Online Presence

Telegram

https://t.me/SolaarProtocol

- 11 218 members
- No active members ⚠️
- Slow response from mods ⚠️


SOLAAR Foundation
32 Tweets

FREE FLOATING RESERVE CURRENCY BOND
BACKED BY THE SOLAAR PROTOCOL

Follow

SOLAAR Foundation
@SolaarProtocol

SOLAAR Protocol | Reserve Currency on #BSC backed by #BTC ₿ , PoL, Crypto Assets and Liquidity Mining; NFT Bonds and Nodes. discord.gg/gyERfDweEs

SolaarProtocol.com Joined November 2021

267 Following 3,449 Followers

Tweets Tweets & replies Media Likes

Pinned Tweet

SOLAAR Foundation @SolaarProtocol · Mar 31

The SOLAAR NFT Bond pre-sale is set to Launch on April 2 @ 6pm UTC / 11am EST on SolaarProtocol.com #DeFiProject #NFTs #NFTdrop #NFTCommunity #DAO

FREE FLOATING RESERVE CURRENCY BOND
BACKED BY THE SOLAAR PROTOCOL

Twitter

https://twitter.com/SolaarProtocol

- 3 437 Followers
- Active

About SpyWolf

SpyWolf is a team of crypto security experts that have been performing full audits for projects for the past months in order to ensure safety on the crypto space. Our goal is to help eliminate monetary fraud through our auditing services and utility token, $SPY.

- Website: SpyWolf.co
- Portal: SpyWolf.network
- Telegram: @SpyWolfNework
- Twitter: Twitter.com/SpyWolfNetwork

(Sample Certificate NFT for those who pass audit)

If you are interested in finding out more about our audits and Certificate of Trust NFTs, reach out to contact@spywolf.co.

Disclaimer

This report shows findings based on our limited project analysis, following good industry practice from the date of this report, in relation to cybersecurity vulnerabilities and issues in the framework and algorithms based on smart contracts, overall social media and website presence and team transparency details of which are set out in this report. In order to get a full view of our analysis, it is crucial for you to read the full report.
While we have done our best in conducting our analysis and producing this report, it is important to note that you should not rely on this report and cannot claim against us on the basis of what it says or doesn't say, or how we produced it, and it is important for you to conduct your own independent investigations before making any decisions. We go into more detail on this in the disclaimer below – please make sure to read it in full.

DISCLAIMER:
By reading this report or any part of it, you agree to the terms of this disclaimer. If you do not agree to the terms, then please immediately cease reading this report, and delete and destroy any and all copies of this report downloaded and/or printed by you. This report is provided for information purposes only and on a non-reliance basis, and does not constitute investment advice.
No one shall have any right to rely on the report or its contents, and SpyWolf and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers and other representatives) (SpyWolf) owe no duty of care towards you or any other person, nor does SpyWolf make any warranty or representation to any person on the accuracy or completeness of the report.
The report is provided "as is", without any conditions, warranties or other terms of any kind except as set out in this disclaimer, and SpyWolf hereby excludes all representations, warranties, conditions and other terms (including, without limitation, the warranties implied by law of satisfactory quality, fitness for purpose and the use of reasonable care and skill) which, but for this clause, might have effect in relation to the report. Except and only to the extent that it is prohibited by law, SpyWolf hereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against SpyWolf, for any amount or kind of loss or damage that may result to you or any other person (including without limitation, any direct, indirect, special, punitive, consequential or pure economic loss or damages, or any loss of income, profits, goodwill, data, contracts, use of money, or business interruption, and whether in delict, tort (including without limitation negligence), contract, breach of statutory duty, misrepresentation (whether innocent or negligent) or otherwise under any claim of any nature whatsoever in any jurisdiction) in any way arising from or connected with this report and the use, inability to use or the results of use of this report, and any reliance on this report. The analysis of the security is purely based on the smart contracts, website, social media and team.
No applications were reviewed for security. No product code has been reviewed.